

Agile-SD: A Linux-based TCP Congestion Control Algorithm for Supporting High-speed and Short-distance Networks

Mohamed A. Alrshah^{1,a}, Mohamed Othman^{2,a}, Borhanuddin Ali^b, Zurina Mohd Hanapi^a

^aDepartment of Communication Technology and Network, Universiti Putra Malaysia, 43400 UPM, Serdang, Selangor D.E., Malaysia

^bDepartment of Computer and Communication Systems Engineering, Universiti Putra Malaysia, 43400 UPM, Serdang, Selangor D.E., Malaysia.

Abstract

Recently, high-speed and short-distance networks are widely deployed and their necessity is rapidly increasing everyday. This type of networks is used in several network applications; such as Local Area Networks (LAN) and Data Center Networks (DCN). In LANs and DCNs, high-speed and short-distance networks are commonly deployed to connect between computing and storage elements in order to provide rapid services. Indeed, the overall performance of such networks is significantly influenced by the Congestion Control Algorithm (CCA) which suffers from the problem of bandwidth under-utilization, especially if the applied buffer regime is very small. In this paper, a novel loss-based CCA tailored for high-speed and Short-Distance (SD) networks, namely Agile-SD, has been proposed. The main contribution of the proposed CCA is to implement the mechanism of agility factor. Further, intensive simulation experiments have been carried out to evaluate the performance of Agile-SD compared to Compound and Cubic which are the default CCAs of the most commonly used operating systems. The results of the simulation experiments show that the proposed CCA outperforms the compared CCAs in terms of average throughput, loss ratio and fairness, especially when a small buffer is applied. Moreover, Agile-SD shows lower sensitivity to the buffer size change and packet error rate variation which increases its efficiency.

Keywords: Agile, CCA, TCP, Linux, High-speed, Short-distance, Bandwidth Utilization, Fairness, Small Buffer.

1. Introduction

In the last decades, the necessity of high-speed and short-distance networks is rapidly increasing everyday due to their wide deployment. Several network applications, such as Local Area Networks (LAN) and Data Center Networks (DCN), are implementing this type of networks (Buyya et al., 2008, Armbrust et al., 2010). These LANs and DCNs serve a very wide range of network-based applications; such as web hosting, searching engines, social media, multimedia broadcasting and storage drives. In the environment of LANs, as shown in Figure 1, and DCNs, as shown in Figure 2 (Al-Fares et al., 2010, Wu and Yang, 2012, Yoo et al., 2012, Prakash et al., 2012), high-speed and short-distance networks are commonly deployed to connect computing

and storage elements to each other in order to provide rapid services. These networks have certain characteristics which are widely different from other types of networks; for instance, link delay is very small which can be a few milliseconds or even hundreds of microseconds and the Bandwidth-Delay-Product (BDP) of the link is very small compared to its equivalent in high-speed and long-distance networks (Tahiliani et al., 2012, Vasudevan et al., 2009).

These attributes could negatively affect the performance of the Transmission Control Protocol (TCP) by making it either more aggressive or more conservative based on the applied approach. In fact, the Congestion Control Algorithm (CCA) is one of the main parts of TCP. It significantly affects the overall performance of such networks, because it is still suffering from the problem of bandwidth under-utilization, especially if the applied buffer regime is very small. This under-utilization of bandwidth is caused by the variation of the aforementioned characteristics of the networks which results either a slow growth of *cwnd* or an over-injection of data into the network (Afanasyev et al., 2010, Scharf,

¹Corresponding authors:

E-mail addresses: mohamed.asnd@gmail.com (Mohamed Alrshah), mothman@upm.edu.my (Mohamed Othman).

²The author is also an associate researcher at the Computational Science and Mathematical Physics Lab, Institute of Mathematical Science, Universiti Putra Malaysia.

2011, Callegari et al., 2012, 2014, Lar and Liao, 2013, Acharya, 2012, Mohamed A. Alrshah et al., 2014).

In order to solve the problem of bandwidth under-utilization over high-speed and Short-Distance (SD) networks, a new loss-based CCA, namely Agile-SD, has been proposed. The main contribution of the proposed CCA is to implement the mechanism of agility factor. Further, intensive simulation experiments have been carried out to evaluate the performance of Agile-SD compared to Compound (the default CCA of MS Windows since Windows Vista) and Cubic (the default CCA of Linux since Kernel 2.6.16) which are the default CCAs of the most commonly used operating systems (Afanasyev et al., 2010, Mohamed A. Alrshah et al., 2014).

The rest of this paper is organized as follows: the related work is presented in Section 2 while Section 3 presents the proposed algorithm “Agile-SD”. Section 4 explains the used approach of performance evaluation which is contains the experiments’ setup, network topology, performance metrics, results and discussion. Finally, Section 5 presents the conclusion and the future work.

2. Related Work

In order to solve the problem of bandwidth under-utilization, Cubic (Ha and Rhee, 2008), Scalable TCP (Kelly, 2003), HS-TCP (Floyd, 2003), BIC (Xu et al., 2004), HCC (Xu et al., 2011), H-TCP (D. Leith, 2004), TCP Africa (King et al., 2005), TCP Compound (Tan and Song, 2006), Fusion (Kaneko et al., 2007), TCP illinois (Liu et al., 2008) and YeAH (Baiocchi et al., 2007) have been developed and implemented in the real operating systems. All of these TCP variants are still unable to fully utilize the available bandwidths of high-speed networks, especially if the used buffer size is less than the BDP of the link (Afanasyev et al., 2010, Scharf, 2011, Callegari et al., 2012, 2014, Lar and Liao, 2013, Acharya, 2012, Mohamed A. Alrshah et al., 2014).

Further, some researchers tried to solve the aforementioned problem by proposing a set of CCAs or TCP variants; such as DCTCP (Alizadeh et al., 2010), ICTCP (Wu et al., 2013), IA-TCP (Hwang et al., 2012) and D²TCP (Vamanan et al., 2012) which designed for data center networks. All of these TCP variants are still suffering from some critical problems, such as the problem of TCP outcast which has not been solved yet (Tahiliani et al., 2012).

Furthermore, some researchers tried to improve the performance of TCP by using parallel approaches; such as AppTCP (Wang et al., 2014), GridFTP (Allcock

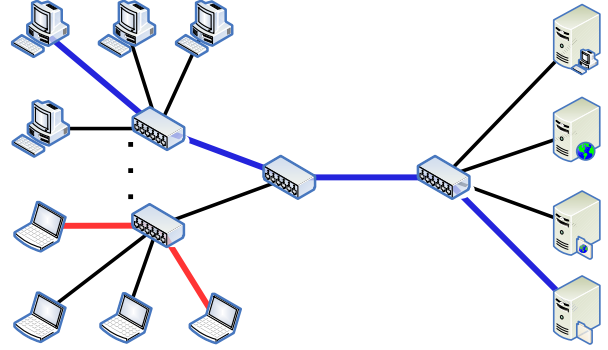


Figure 1: A Network Topology for LAN.

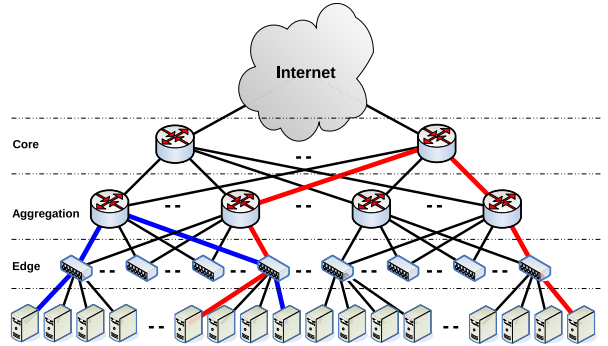


Figure 2: Multi-rooted Hierarchical Topology of Data Centers.

et al., 2005), pTCP (Hsieh and Sivakumar, 2002), BBCP (Hanushevsky et al., 2001), PSocket (Sivakumar et al., 2000), MulTCP (Crowcroft and Oechslin, 1998), DPSS (Tierney et al., 1994) and Parallel-TCP (Mohamed A. Alrshah and Mohamed Othman, 2009, Mohamed A. Alrshah and Mohamed Othman, 2013). Most of parallel schemes have achieved high bandwidth utilization, but unfortunately they have another issues which limited their deployments. One of these issues is that all of the parallel schemes have a very high aggressiveness level compared to the existing single-based TCP variants. This aggressive behavior negatively affects the fairness (Fu and Indulska, 2005, Fu et al., 2007).

However, the widely deployed CCAs; such as Compound and Cubic which have been set as the default CCAs in MS Windows and Linux operating systems, respectively; are playing the important role in the real networks. Thus, Compound and Cubic should be used as a benchmark, to confirm the performance of any newly proposed CCA or TCP variant. For this reason, these two CCAs are going to be briefly explained in the next subsections.

2.1. Compound TCP (C-TCP)

The widely deployed TCP variant namely C-TCP (Tan and Song, 2006) is the default CCA of MS Windows since Windows Vista. C-TCP combines HS-TCP (Floyd, 2003) and NewReno (Floyd and Henderson, 1999) to be used as fast and slow modes, respectively. C-TCP is a loss-delay-based approach relies on multi-modes switching to increase the bandwidth utilization over high-speed networks. Generally, it improves the performance of TCP to some extent but it introduces another problem which is the RTT mis-estimation. This problem has been inherited from Vegas (Brakmo and Peterson, 1995) and it can negatively affect the overall performance of the protocol (Afanasyev et al., 2010, Mohamed A. Alrshah et al., 2014).

2.2. CUBIC TCP (Cubic)

Cubic (Ha and Rhee, 2008) is the default CCA of Linux operating systems since its implementation in Kernel 2.6.16. Cubic enhances the bandwidth utilization over high-speed networks by increasing the *cwnd* in the congestion avoidance phase by a *cubic* function of the elapsed time since last loss. In addition, Cubic forces its *cwnd* not to be less than the pre-calculated *cwnd* of NewReno. Despite of all, Cubic is still suffering from the under-utilization of high-speed bandwidth specifically when the used buffer size is small (Afanasyev et al., 2010, Mohamed A. Alrshah et al., 2014, Ha and Rhee, 2008).

2.3. The Latest Issues

Recent studies have revealed that all of the current TCP variants have different levels of inability on fully utilizing the bandwidths over the new generation of high-speed networks, especially if a *near-zero* buffer is applied. Thus, it becomes very necessary to design a new CCA to increase the bandwidth utilization over such networks (Afanasyev et al., 2010, Mohamed A. Alrshah et al., 2014).

3. Agile-SD: The Proposed Algorithm

Algorithm 1 explains the Agile-SD mechanism which is geared to work on high-speed and short-distance networks to enhance the overall performance and bandwidth utilization while preserving the fairness. Moreover, Figure 3 shows the flow control diagram of Agile-SD and the following subsections explain the proposed algorithm in more details.

Algorithm 1: Agile-SD Congestion Avoidance.

```

1 Initialization:
2    $\lambda_{min} \leftarrow 1, \lambda_{max} \leftarrow 3,$ 
3    $\beta_1 \leftarrow 0.90, \beta_2 \leftarrow 0.95,$ 
4    $cwnd \leftarrow 2$ 
5 Event On ACK Reception do
6   calculate  $gap_{current}$  as in Equation (3)
7   calculate  $gap_{total}$  as in Equation (2)
8   calculate  $\lambda$  as in Equation (1)
9    $\alpha = \frac{\lambda}{cwnd}$ 
10   $cwnd \leftarrow cwnd + \alpha$ 
11 end
12 Event On Loss Detection of 3-duplicated ACKs do
13   $cwnd_{loss} \leftarrow cwnd$ 
14  if  $tcp\_status = SlowStart$  then
15     $cwnd \leftarrow cwnd \times \beta_1$ 
16  else
17     $cwnd \leftarrow cwnd \times \beta_2$ 
18  end
19   $ssthresh \leftarrow cwnd - 1$ 
20   $cwnd_{degraded} \leftarrow cwnd$ 
21 end

```

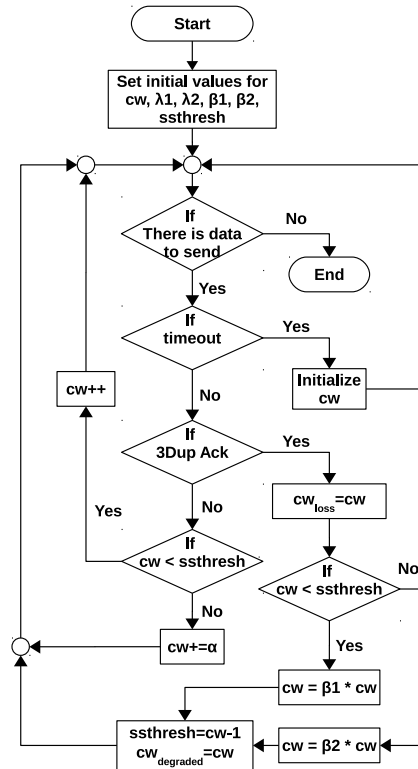


Figure 3: The flow control diagram of Agile-SD.

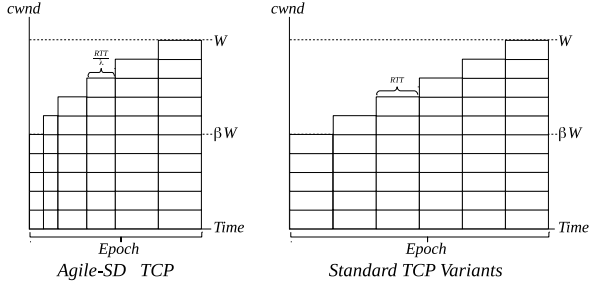


Figure 4: The $cwnd$ evolution of Agile-SD and the standard TCP.

3.1. The Agility Factor Mechanism

As known, Chu et al. (2013) increases the initial value of TCP $cwnd$ to 10 packets, but Agile-SD initializes its $cwnd$ by 2 packets in order to focus on the impact of CA on bandwidth utilization. However, in the future implementations the initial $cwnd$ should be set to 10 to gain better bandwidth utilization.

Clearly, Agile-SD increases its $cwnd$ in the stage of congestion avoidance by fraction similarly as the existing CCAs. But, Agile-SD increases its $cwnd$ by $\frac{\lambda}{cwnd}$ to show a convex curve unlike the standard TCP which increases its $cwnd$ linearly by $\frac{1}{cwnd}$. The main contribution of Agile-SD is the unique $cwnd$ growth function which relies on the agility factor mechanism which symbolized by λ , as shown in Equation (1).

$$\lambda = \max\left(\frac{\lambda_{max} \times gap_{current}}{gap_{total}}, \lambda_{min}\right) \quad (1)$$

where, gap_{total} exemplifies the amount of decrease in the $cwnd$ which caused by the loss event. In other words, gap_{total} represents the released amount from the bandwidth after loss. gap_{total} is calculated as the distance between the maximum recorded limit of the bandwidth ($cwnd_{loss}$) and the $cwnd_{degraded}$ as in Equation (2).

$$gap_{total} = \max((cwnd_{loss} - cwnd_{degraded}), 1) \quad (2)$$

While, $gap_{current}$ is calculated as the difference between the maximum recorded limit of the bandwidth ($cwnd_{loss}$) and the current $cwnd$, as in Equation (3).

$$gap_{current} = \max((cwnd_{loss} - cwnd), 1) \quad (3)$$

Simply, λ is used to mitigate the impact of loss degradation on the overall performance of TCP. Specifically, λ shortens the time of epoch which is needed by CCA to move its $cwnd$ from $cwnd_{degraded}$ to $cwnd_{loss}$ or to the maximum allowed $cwnd$, as shown in Figure 4. In order to increase the bandwidth utilization, λ speeds up the

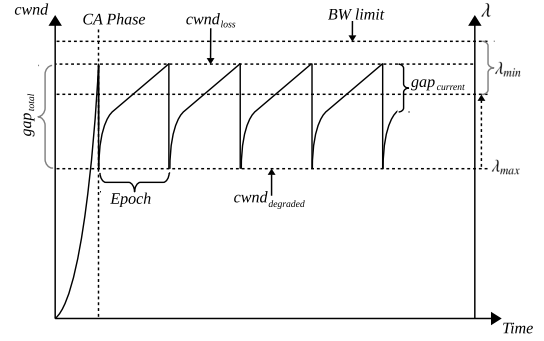


Figure 5: The concept of agility factor mechanism λ .

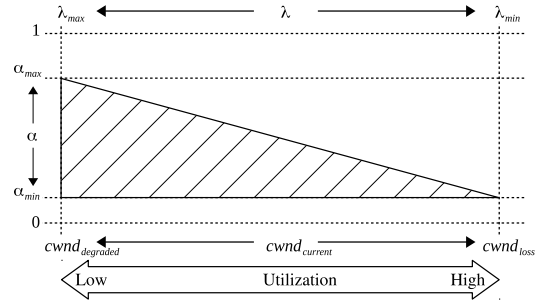


Figure 6: Relation between α , λ and the bandwidth utilization.

growth of $cwnd$ when the current $cwnd$ is far away from the last $cwnd_{loss}$. While, it conservatively slows down the growth of $cwnd$ when the current $cwnd$ nearing to the last $cwnd_{loss}$.

More specifically, to ensure that the performance of Agile-SD is not less than the standard TCP, λ_{min} must be always set to 1 while λ_{max} must be always set to a value ≥ 1 . However, if λ_{max} is set to 1, Agile-SD will behave exactly similar to NewReno. But, if it was set to a value > 1 , such as 2, 3 or 4, it would clearly improve the overall performance.

After loss detection by receiving three duplicate ACKs, Agile-SD initiates its agility factor by λ_{max} , then reduces it every cycle towards λ_{min} . Nevertheless, it restarts by λ_{max} again if another three duplicate ACKs are received as shown in Figure 5. Consequently, it is very clear that α is directly proportional to λ , and λ is reversely proportional to the current size of $cwnd$, as shown in Figure 6. In other words, the aggressiveness of the proposed algorithm increases whenever the difference between $cwnd$ and $cwnd_{loss}$ increases, as in Equation (4).

$$\alpha_{max} = \lim_{gap_{current} \rightarrow gap_{total}} \frac{\max\left(\frac{\lambda_{max} \times gap_{current}}{gap_{total}}, \lambda_{min}\right)}{cwnd} \quad (4)$$

$$= \frac{\lambda_{max}}{cwnd}$$

Contrarily, it decreases its aggressiveness whenever the utilization is around to touch the maximum available bandwidth, as in Equation (5).

$$\alpha_{min} = \lim_{gap_{current} \rightarrow zero} \frac{\max\left(\frac{\lambda_{max} \times gap_{current}}{gap_{total}}, \lambda_{min}\right)}{cwnd} \quad (5)$$

$$= \frac{\lambda_{min}}{cwnd}$$

In general, Agile-SD reduces the cycle time which by its role reduces the epoch time to overcome the problem of slow evolution of $cwnd$ provided by the standard TCP, as shown early in Figure 4. On one hand, this behavior guarantees the performance of Agile-SD not to be lower than the standard TCP. Thus, it increases the bandwidth utilization by improving the ability of Agile-SD to expose the condition of the underlying network as shown in Figure 6. On the other hand, it reduces the sensitivity of Agile-SD to the loss rate.

3.2. The Decrement of $cwnd$

The standard TCP applies the *Multiplicative Decrease* mechanism which halves the $cwnd$ after any loss detection, by receiving three duplicate ACKs, regardless of which stage the loss is detected in. Unlikely, Agile-SD decreases its $cwnd$ after any loss detection by two ways based on the stage which the loss is coming from. First, if the loss is detected in the slow start stage, Agile-SD reduces its $cwnd$ to $\beta_1\%$ of the latest $cwnd$ as shown at Line 15 in Algorithm 1. Second, if the loss is detected in the congestion avoidance stage, Agile-SD reduces its $cwnd$ to $\beta_2\%$ of the latest $cwnd$ as shown at Line 17 in Algorithm 1. Moreover, Agile-SD sets the $ssthresh$ to $cwnd - 1$, after any degradation, in order to avoid slipping into an undesirable slow start.

Since, the loss which happens in the slow start stage is more severe than which happens in the congestion avoidance stage. Therefore, the value of β_1 should be always less than β_2 . In other words, the reduction which follows a slow start loss should be greater than the reduction which follows a congestion avoidance loss. Also, β_1 and β_2 must be reversely proportional to their relative λ_{max} but the relation among them is still under investigation to be revealed in the future work. Thus, whenever the values of β_1 and β_2 is increased, the value

of λ_{max} should be adaptively decreased and vice versa. For instance, if β_1 and β_2 are set to 0.9 and 0.95, respectively, where these values are compatible with $\lambda_{max} = 3$. Then, if β_1 and β_2 are reduced to 0.85 and 0.9, respectively, the value of λ_{max} should be increased to a value, such as 4 or 5, and so on.

3.3. Agile-SD Overall Behavior

Similar to standard TCP, Agile-SD starts by slow start to show an exponential increase until the detection of first loss; by receiving three duplicate ACKs. This reduces its $cwnd$ to $\beta_1\%$ and triggers the congestion avoidance function. In this stage, Agile-SD increases its $cwnd$ by α to show a convex curve. But, if the $cwnd$ becomes closer to the bandwidth limit which is set as $cwnd_{loss}$, it starts a linear increase until detecting another packet loss. If the event of packet loss is detected, Agile-SD reduces its $cwnd$ to $\beta_2\%$ then repeats the same stages which follows the slow start stage. However, if timeout is detected at any stage, Agile-SD resets its $cwnd$ to the initial value, as shown in Figure 3.

For more understanding, assume that there is a TCP link with $cwnd_{loss} = 12$, $cwnd_{degraded} = 9$ and a constant RTT equal to 20 ms, and the congestion avoidance stage is just started after the loss directly. Thus, the number of cycles needed by any CCA to reach $cwnd_{loss}$ is 4 cycles which is equal to $(cwnd_{loss} - cwnd_{degraded} + 1)$. Consequently, the epoch time needed by the standard TCP, which is " RTT - dependent", is the number of needed cycles times RTT , so it will be equal to 80 ms.

Instead, Agile-SD increases its $cwnd$ independently from the RTT . Thus, every cycle consumes a time of $\frac{RTT}{\lambda}$ to send a number of $\frac{cwnd}{\lambda}$ packets during that cycle, then it increases its $cwnd$ by 1. Consequently, the epoch time needed by Agile-SD will be equal to what shown in Equation (6).

$$EpochTime = \sum_{i=1}^k \frac{RTT}{\lambda_i} \quad (6)$$

where k is the number of needed cycles.

Suppose λ_{min} and λ_{max} are set to 1 and 4, respectively. So, λ_i will take the value of [4, 3, 2, 1] sequentially, which will result in an epoch time equal to 41.66 ms. Thus, the epoch time of Agile-SD will be shrunk by around 48% from the epoch time of the standard TCP on the same network link. This behavior helps Agile-SD to increase its $cwnd$ more quickly than the other compared CCAs and consequently improves the bandwidth utilization. In other words, the faster $cwnd$ growth is the higher bandwidth utilization and vice versa.

4. Performance Evaluation of Agile-SD

The goal of this work is, to develop a new CCA, namely Agile-SD, which has the ability of increasing the bandwidth utilization over high-speed and short-distance networks while maintaining fairness. Agile-SD CCA has been implemented as a pluggable Linux CCA module which can be plugged into any Linux Kernel. As well as, this module has the ability to be plugged into NS-2 network simulator, as a Linux TCP, in order to evaluate its performance compared to some of the widely deployed CCAs.

4.1. The Experiments Setup

In this work, intensive simulation experiments have been conducted using the well-known network simulator NS-2 version 2.35, to evaluate the proposed CCA by comparing its performance with C-TCP and Cubic. The conducted experiments have been divided into three main scenarios: single-flow, sequentially established/terminated multiple-flows, and synchronously established/terminated multiple-flows. Table 1 shows the setting of the experiments' parameters as used in this work.

Table 1: Experiment Parameters.

No. Parameter	Value
1. CCAs	Agile-SD, Cubic, C-TCP
2. Link capacity	1 Gbps for all
3. Link delay	1ms (node to router) 4ms (router to router)
4. BDP	750KB (As in (Jacobson and Braden, 1988))
5. PER	zero, 10^{-5} , 10^{-4}
6. Buffer size	from 5 to 500 packets
7. Packet size	1000 bytes
8. Queuing Algo	Drop-Tail
9. Traffic type	FTP
10. SACK, FACK	Disabled
11. Simulation time	100 seconds

In the first scenario of single-flow, there is only one pair of sender and receiver, as shown in Figure 7, which presents an ideal case with no congestion to show the ability of the evaluated CCAs on achieving full bandwidth utilization. As for the second and third scenarios of multiple flows, there are n pairs of sender and receiver, as shown in Figure 8, which have been used

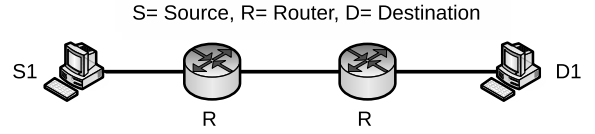


Figure 7: Non-congested Network topology.

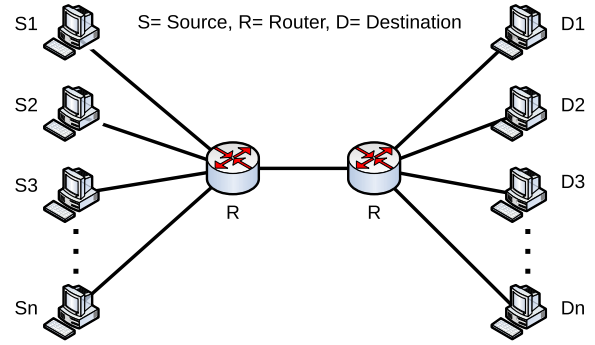


Figure 8: Network topology with standard dumbbell bottleneck.

to simulate the network congestion and to show its impact on the performance measurements of the evaluated CCAs. In the second scenario, the flows are sequentially established and terminated as shown in Figure 9(a). While in the third scenario, the flows are synchronously established and terminated as shown in Figure 9(b).

In all of these experiments, a standard single dumbbell topology has been used, as shown in figures 7 and 8, where n is the competing senders ($S1, S2, S3, \dots, Sn$) which send data simultaneously to n receivers ($D1, D2, D3, \dots, Dn$) through the shared single bottleneck. All source and destination nodes are connected to the bottleneck routers over LAN with $1Gbps$ speed and $1ms$ propagation delay. While the bottleneck link is $1Gbps$ speed with a propagation delay of $4ms$ (Wang et al., 2014). These experiments have been repeated for each CCA separately with variable buffer size and variable packet error rate (PER). The buffer size varies from 5 to 500 packets while the PERs which have been used are 10^{-4} , 10^{-5} and zero PER.

Moreover, the performance metrics evaluated in this paper are the average throughput, loss ratio, inter-fairness, intra-fairness and RTT-fairness. Average throughput and loss ratio are evaluated to reflect the ability of the TCP variant on utilizing the bandwidth. While, measuring inter-fairness, intra-fairness and RTT-fairness is to show the quality of sharing the link between the competing TCP flows based on Jain's fairness index (JFI) (Jain et al., 1984), as shown in Equation (7).

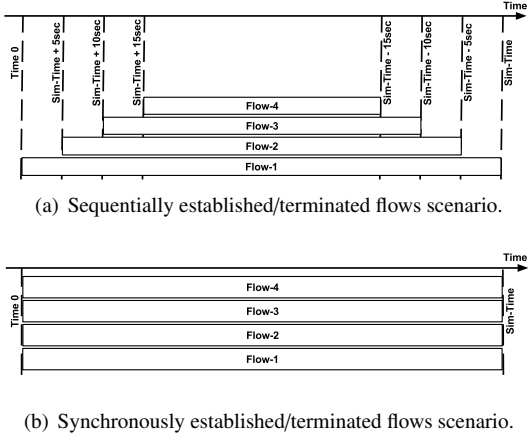


Figure 9: The sequence of establishments and terminations of the multiple flows scenarios.

$$JFI(x_1, x_2, \dots, x_n) = \frac{(\sum_{i=1}^n x_i)^2}{n \cdot \sum_{i=1}^n x_i^2} \quad (7)$$

Substantially, these experiments show the impact of bottleneck congestion, buffer size and PER on the performance of the examined CCAs and also show the performance changes when a smaller buffer size is applied. Moreover, the simulation time used in all experiments has been set to 100 seconds which is enough for TCP to show its steady state.

4.2. Results and Discussion

This subsection presents an analytical discussion of the behavior exhibited by the proposed CCA and the compared CCAs. As well as, it presents the results of the performance evaluation and shows the measurements of the average throughput, loss ratio, inter-fairness, intra-fairness and RTT-fairness.

4.2.1. The *cwnd* evolution

Fundamentally, the target of CCAs is: to maximize the throughput while minimizing the loss ratio and maintaining the fairness. Figure 10 shows the *cwnd* evolution of the studied CCAs based on the buffer size change. Due to the mechanism of agility factor, Agile-SD expectedly shows the faster *cwnd* growth followed by Cubic and C-TCP. This fast or slow evolution of *cwnd* is the core of any CCA which would directly affect the other performance metrics, such as throughput, loss ratio and it may affect the fairness as well.

In Figure 10(a), it is very clear that Agile-SD reaches the maximum *cwnd*, which is about 1500 packets, in around 17 second then starts oscillating to show very short epochs, while, Cubic reaches the maximum *cwnd* in about 60 seconds then starts oscillating to draw very long epochs. As for C-TCP, it fails to reach the maximum *cwnd* and touches only the edge of 110 packets then exhibits very short epochs. Indeed, the larger the *cwnd*, the higher the throughput and vice versa.

Interestingly, when the buffer size increases, the behavior of the studied CCAs relatively improves. The figures from 10(b) to 10(e) show that Agile-SD and Cubic reduce their time of reaching the maximum *cwnd* whenever the buffer size increases. Besides, Agile-SD keeps showing short epochs while Cubic remains drawing long epochs. Unlikely, C-TCP heightens its *cwnd* towards the maximum limit whenever the buffer size increases. In fact, the wider oscillating and/or the longer epochs is the lower bandwidth utilization and vice versa. Thus, the higher bandwidth utilization among the studied CCAs would be provided by Agile-SD followed by Cubic then C-TCP.

4.2.2. The average throughput

In the first scenario, as shown in Figure 11(a), Agile-SD has overcome the other CCAs in terms of average throughput due to its fast growth of *cwnd* resulted by the mechanism of agility factor. Moreover, Agile-SD presents lower sensitivity to PER than the others, whereas, Cubic and C-TCP are highly affected by PER and they present a poor performance when the PER is increased. However, in the cases of 10^{-4} and 10^{-5} PER as shown in figures 11(b) and 11(c), C-TCP presents better performance than Cubic in most cases. In general, Agile-SD achieves better throughput than the others in most cases even in the lossy environments. Clearly, it improves the bandwidth utilization up to 55% in some cases of this scenario.

For the second scenario, the figures 11(d), 11(e) and 11(f) show that Agile-SD has overcome the other CCAs, in term of average throughput, at most cases even when the buffer size is small and the PER is high. Furthermore, it improves the bandwidth utilization from 10% to 40% in the cases of 5 packets buffer size. While in the third scenario, Agile-SD has outperformed the other CCAs in all cases especially when a *near-zero* buffer is applied as shown in figures 11(g), 11(h) and 11(i). Moreover, Agile-SD significantly improves the bandwidth utilization even when the PER is high. Thus, it provides up to 40% of improvement in some cases.

4.2.3. The loss ratio

As regarding to all scenarios, the studied CCAs have presented a loss ratio lower than 0.5% which is considered as a negligible loss ratio. Thus, Figure 12 has been selected here as a sample of the loss ratio results of all scenarios to save the space of this paper and because the rest of the figures have no much difference.

4.2.4. The fairness

As for intra-fairness and RTT-fairness, all the studied CCAs are interchangeably close to each other. However, in some cases Agile-SD seems more fair, especially when the applied buffer size is small. Since the difference among the graphs of the results is very trivial, the figures 13 and 14 have been chosen as samples of intra-fairness and RTT-fairness, respectively.

Moreover, a separated experiment has been carried out to evaluate the inter-fairness among the studied CCAs and the standard NewReno, using the same topology as shown in Figure 8, where the result is shown in Figure 15. For inter-fairness to NewReno, Agile-SD and C-TCP achieve around 0.76 while Cubic achieves about 0.79 inter-fairness index. As for the inter-fairness to Cubic, Agile-SD scores the highest index which is almost 0.99 and C-TCP scores around 0.96 while NewReno achieves only 0.79 inter-fairness index. As for the inter-fairness to C-TCP, Cubic scores the highest index which is around 0.96 while Agile-SD and NewReno achieve about 0.76 inter-fairness index.

5. Conclusion

In this paper, a new CCA, namely Agile-SD, has been proposed and evaluated. The main contribution of the proposed CCA is to implement the mechanism of agility factor. The need of the proposed CCA has been arisen by the inability of the existing high-speed CCAs in achieving a full bandwidth utilization over high-speed networks, especially when a small buffer regime is applied. Further, a new CCA module has been implemented and plugged into the Linux kernel version 3.19.0. As well as, this module has been plugged into the Network Simulator NS-2 version 2.35, as a Linux TCP, in order to evaluate it by comparing its performance to the other CCAs.

Subsequently, intensive simulation experiments have been conducted to evaluate the proposed CCA by comparing its performance to C-TCP and Cubic, which are the current default TCP algorithms of MS Windows and Linux, respectively. The results show that the proposed algorithm achieves higher bandwidth utilization

than the existing CCAs while maintaining fairness. Due to the use of agility factor, Agile-SD shows lower sensitivity to the changes of buffer size and PER.

Importantly, Agile-SD presents higher performance than the compared CCAs and it provides a significant improvement which is: up to 55% in the case of single flow, up to 40% in the case of sequentially established/terminated multi-flows and up to 40% in the case of synchronously established/terminated multi-flows.

More importantly, the second scenario presents the real case of network, in which all TCP flows are not established or terminated synchronously. In this scenario, Agile-SD has achieved up to 95% bandwidth utilization while the others did not exceed it in the case of large buffer. As for the case of small buffer, Agile-SD achieves around 92% bandwidth utilization while the other TCP variants achieve from 32% to 85% bandwidth utilization.

Eventually, Agile-SD is a sender-side TCP module which does not change anything at receiver-side. It uses the standard slow start and provides a new congestion avoidance algorithm featured by the mechanism of agility factor. Currently, we have already implemented Agile-SD into the latest Linux kernel 3.19.0 and a real dumbbell topology has been built using Dummynet over PC-BSD version 10 to evaluate the proposed CCA based on real test-bed in the nearest future. Also, there is a strong intention to evaluate Agile-SD with SACK and/or FACK features to show their impacts on the throughput. As well as, Agile-SD should have the ability to consider the delayed acknowledgments which needs some modification at the receiver-side.

Acknowledgments

This work has been partially supported by the Malaysian Ministry of Education under the Fundamental Research Grant FRGS/02/01/12/1143/FR for financial support.

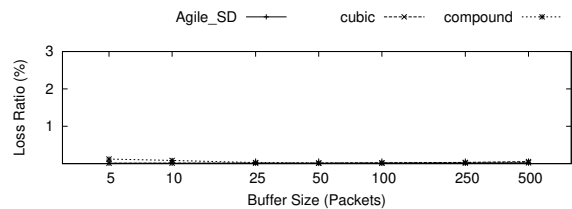


Figure 12: 1st Scenario (10^{-4} PER): Loss Ratio vs. Buffer Size.

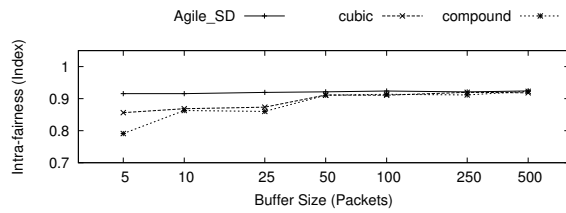


Figure 13: 2nd Scenario (Zero PER): Intra-Fairness vs. Buffer Size.

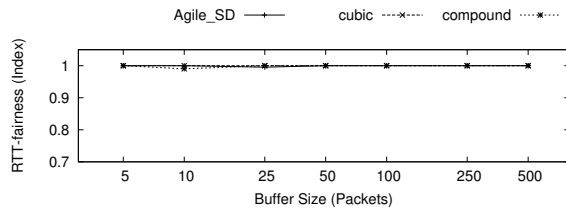


Figure 14: 3rd Scenario (Zero PER): RTT-Fairness vs. Buffer Size.

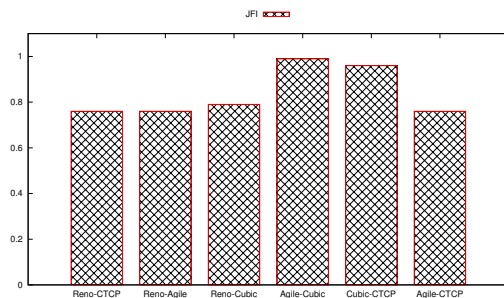


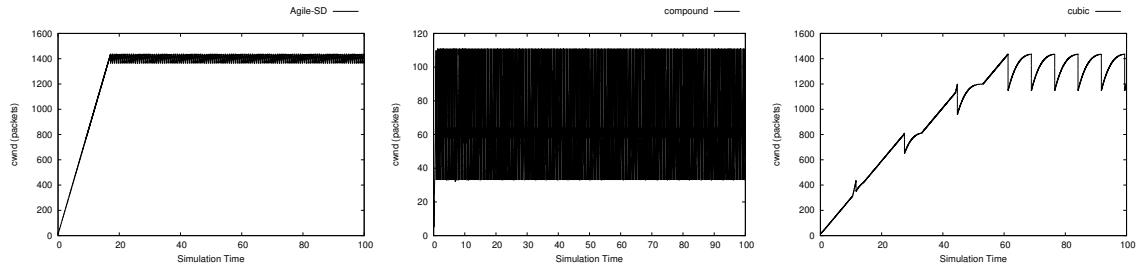
Figure 15: The inter-fairness among the studied CCAs.

References

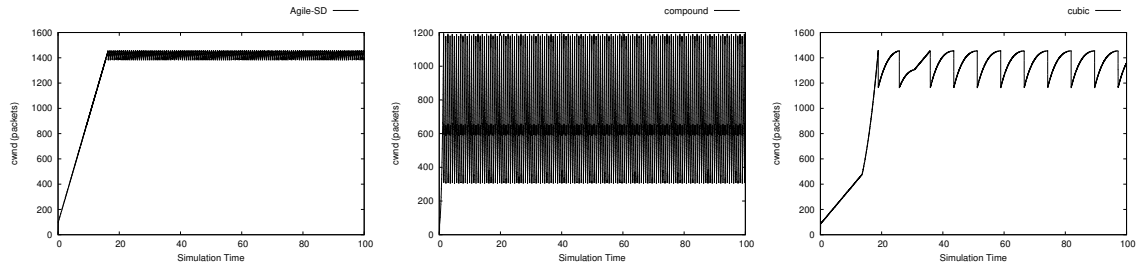
- Acharya, S., 2012. Study and analysis of tcp/ip congestion control techniques: A review. Illinois Journalism Education Association. .
- Afanashev, A., Tilley, N., Reiher, P., Kleinrock, L., 2010. Host-to-Host Congestion Control for TCP. IEEE Communications Surveys and Tutorials 12 (3), 304–342.
- Al-Fares, M., Radhakrishnan, S., Raghavan, B., Huang, N., Vahdat, A., 2010. Hedera: Dynamic flow scheduling for data center networks. In: NSDI. Vol. 10. pp. 19–19.
- Alizadeh, M., Greenberg, A., Maltz, D. A., Padhye, J., Patel, P., Prabhakar, B., Sengupta, S., Sridharan, M., 2010. Data center tcp (dctcp). In: Proceedings of the ACM SIGCOMM 2010 Conference, SIGCOMM '10. ACM, New York, NY, USA, pp. 63–74.
- Allcock, W., Bresnahan, J., Kettimuthu, R., Link, M., Dumitrescu, C., Raicu, I., Foster, I., 2005. The globus striped gridftp framework and server. In: Proceedings of the 2005 ACM/IEEE conference on Supercomputing. Vol. 1. IEEE Computer Society, pp. 1–11.
- Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M., Apr. 2010. View of cloud computing. Comm. ACM 53 (4), 50–58.
- Baiocchi, A., Castellani, A. P., Vacirca, F., 2007. YeAH-TCP: Yet Another Highspeed TCP. In: Proc. PFLDnet. Roma, Italy, pp. 37–42.
- Brakmo, L. S., Peterson, L. L., 1995. Tcp vegas: End to end congestion avoidance on a global internet. Selected Areas in Communications, IEEE Journal on 13 (8), 1465–1480.

- Buyya, R., Yeo, C. S., Venugopal, S., Sept 2008. Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. In: 10th IEEE International Conference on High Performance Computing and Communications, 2008. pp. 5–13.
- Callegari, C., Giordano, S., Pagano, M., Pepe, T., 2012. Behavior analysis of TCP Linux variants. Computer Networks 56 (1), 462–476.
- Callegari, C., Giordano, S., Pagano, M., Pepe, T., 2014. A survey of congestion control mechanisms in linux tcp. In: Distributed Computer and Communication Networks. Vol. 279 of Communications in Computer and Information Science. Springer, pp. 28–42.
- Chu, J., Cheng, Y., Dukkupati, N., Mathis, M., April 2013. Increasing tcp's initial window. RFC 6928, IETF Network Working Group. .
- Crowcroft, J., Oechslin, P., 1998. Differentiated end-to-end internet services using a weighted proportional fair sharing tcp. ACM SIGCOMM Computer Communication Review 28 (3), 53–69.
- D. Leith, R. S., 2004. H-tcp: Tcp for high-speed and long distance networks. In: Proceedings of PFLDnet. pp. 95–101.
- Floyd, S., April 2003. HighSpeed TCP for Large Congestion Windows. RFC 3649, IETF Network Working Group. .
- Floyd, S., Henderson, T., April 1999. The newreno modification to tcps fast recovery algorithm. RFC 2582, IETF Network Group. .
- Fu, Q., Indulski, J., 2005. Features of parallel tcp with emphasis on congestion avoidance in heterogeneous networks. In: Wysocki, T., Dadej, A., Wysocki, B. (Eds.), Advanced Wired and Wireless Networks. Vol. 26 of Multimedia Systems and Applications Series. Springer US, pp. 207–230.
- Fu, Q., Indulski, J., Perreau, S., Zhang, L., 2007. Exploring tcp parallelisation for performance improvement in heterogeneous networks. Computer Communications 30 (17), 3321–3334.
- Ha, S., Rhee, I., 2008. CUBIC: A New TCP-Friendly High-Speed TCP Variant. SIGOPS Operating Systems Review 42 (5), 64–74.
- Hanushevsky, A., Trunov, A., Cottrell, L., 2001. Peer-to-peer computing for secure high performance data copying. In: In Proc. of the 2001 Int. Conf. on Computing in High Energy and Nuclear Physics (CHEP 2001), Beijing. Vol. 2001. Citeseer.
- Hsieh, H.-Y., Sivakumar, R., 2002. ptcp: An end-to-end transport layer protocol for striped connections. In: 10th IEEE International Conference on Network Protocols Proceedings. Vol. 2002. IEEE, pp. 24–33.
- Hwang, J., Yoo, J., Choi, N., June 2012. Ia-tcp: a rate based incast-avoidance algorithm for tcp in data center networks. In: IEEE International Conference on Communications (ICC). pp. 1292–1296.
- Jacobson, V., Braden, R., October 1988. Tcp extensions for long-delay paths. RFC 1072, IETF Network Working Group. .
- Jain, R., Chiu, D.-M., Hawe, W. R., 1984. A quantitative measure of fairness and discrimination for resource allocation in shared computer system. Eastern Research Laboratory, Digital Equipment Corporation.
- Kaneko, K., Fujikawa, T., Su, Z., Katto, J., 2007. TCP-Fusion : A Hybrid Congestion Control Algorithm for High-speed Networks. In: Proc. PFLDnet, ISI, Marina Del Rey, California. pp. 31–36.
- Kelly, T., 2003. Scalable TCP : Improving Performance in Highspeed Wide Area Networks. ACM SIGCOMM Computer Communications Review 33 (2), 83–91.
- King, R., Baraniuk, R., Riedi, R., 2005. TCP-Africa: An adaptive and fair rapid increase rule for scalable TCP. In: INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE. pp. 1–11.
- Lar, S.-u., Liao, X., 2013. An initiative for a classified bibliography on tcp/ip congestion control. Journal of Network and Computer Applications 36 (1), 126–133.
- Liu, S., Başar, T., Srikant, R., 2008. Tcp-illinois: A loss-and delay-based congestion control algorithm for high-speed networks. Per-

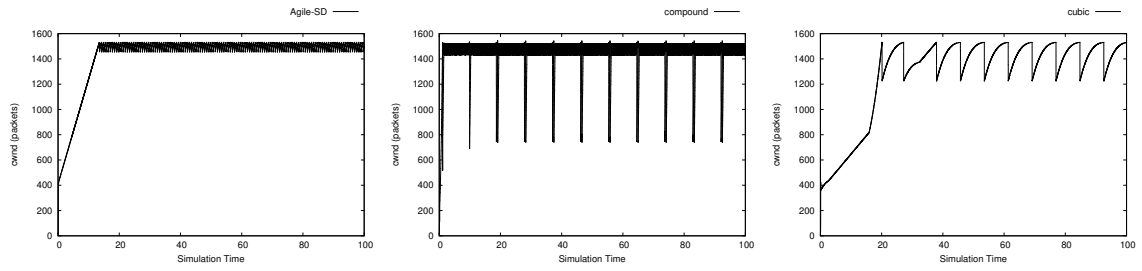
- formance Evaluation 65 (6), 417–440.
- Mohamed A. Alrshah, Mohamed Othman, Nov 2009. Test-bed based comparison of single and parallel tcp and the impact of parallelism on throughput and fairness in heterogenous networks. In: ICCTD '09. International Conference on Computer Technology and Development, 2009. Vol. 1. pp. 332–335.
- Mohamed A. Alrshah, Mohamed Othman, Borhanuddin Ali, Zurina Mohd Hanapi, 2014. Comparative study of high-speed linux tcp variants over high-BDP networks. *Journal of Network and Computer Applications* 43, 66–75.
- Mohamed A. Alrshah and Mohamed Othman, Nov 2013. Performance evaluation of parallel TCP, and its impact on bandwidth utilization and fairness in High-BDP networks based on test-bed. In: 2013 IEEE Malaysia International Conference on Communications (MICC). pp. 23–28.
- Prakash, P., Dixit, A., Hu, Y. C., Kompella, R., 2012. The tcp outcast problem: Exposing unfairness in data center networks. In: Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation, San Jose, CA. p. .
- Scharf, M., 2011. Comparison of end-to-end and network-supported fast startup congestion control schemes. *Computer Networks* 55 (8), 1921–1940.
- Sivakumar, H., Bailey, S., Grossman, R. L., 2000. Psockets: The case for application-level network striping for data intensive applications using high speed wide area networks. In: Proceedings of the 2000 ACM/IEEE conference on Supercomputing. Vol. 2000. IEEE Computer Society, pp. 1–7.
- Tahiliani, R., Tahiliani, M., Sekaran, K., Dec 2012. Tcp variants for data center networks: A comparative study. In: International Symposium on Cloud and Services Computing (ISCOS). pp. 57–62.
- Tan, K., Song, J., 2006. Compound tcp: A scalable and tcp-friendly congestion control for high-speed networks. In: in 4th International workshop on Protocols for Fast Long-Distance Networks (PFLDNet), 2006. pp. 80–83.
- Tierney, B., Lee, J., Chen, L. T., Herzog, H., Hoo, G., Jin, G., Johnston, W. E., 1994. Distributed parallel data storage systems: a scalable approach to high speed image servers. In: Proceedings of the second ACM international conference on Multimedia. Vol. 1994 of MULTIMEDIA '94. ACM, pp. 399–405.
- Vamanan, B., Hasan, J., Vijaykumar, T., 2012. Deadline-aware data-center tcp (d2tcp). In: Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication. SIGCOMM '12. ACM, New York, NY, USA, pp. 115–126.
- Vasudevan, V., Phanishayee, A., Shah, H., Krevat, E., Andersen, D. G., Ganger, G. R., Gibson, G. A., Mueller, B., Aug. 2009. Safe and effective fine-grained tcp retransmissions for datacenter communication. *SIGCOMM Comput. Commun. Rev.* 39 (4), 303–314.
- Wang, G., Wu, Y., Dou, K., Ren, Y., Li, J., 2014. Apptcp: The design and evaluation of application-based tcp for e-vlbi in fast long distance networks. *Future Generation Computer Systems* 39, 67–74.
- Wu, H., Feng, Z., Guo, C., Zhang, Y., April 2013. Ictcp: Incast congestion control for tcp in data-center networks. *Networking, IEEE/ACM Transactions on* 21 (2), 345–358.
- Wu, X., Yang, X., June 2012. Dard: Distributed adaptive routing for datacenter networks. In: 2012 IEEE 32nd International Conference on Distributed Computing Systems (ICDCS). pp. 32–41.
- Xu, L., Harfoush, K., Rhee, I., 2004. Binary increase congestion control (bic) for fast long-distance networks. In: INFOCOM 2004. Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies. Vol. 4. pp. 2514–2524.
- Xu, W., Zhou, Z., Pham, D., Ji, C., Yang, M., Liu, Q., 2011. Hybrid congestion control for high-speed networks. *Journal of Network and Computer Applications* 34 (4), 1416–1428.
- Yoo, S. J. B., Yin, Y., Wen, K., April 2012. Intra and inter datacenter networking: The role of optical packet switching and flexible bandwidth optical networking. In: Optical Network Design and Modeling, 2012 16th International Conference on. pp. 1–6.



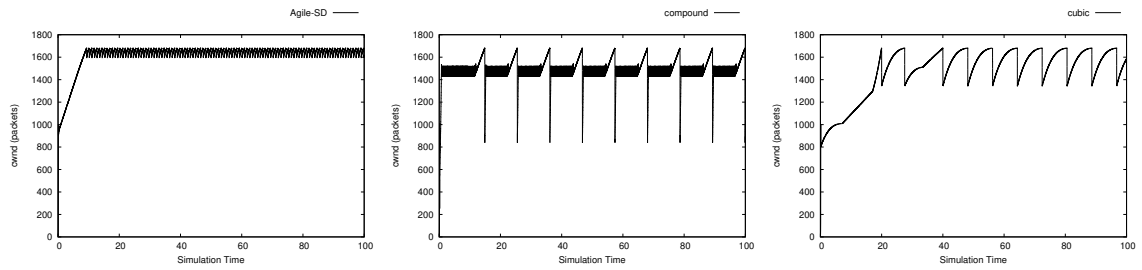
(a) Buffer Size = 5 Packets.



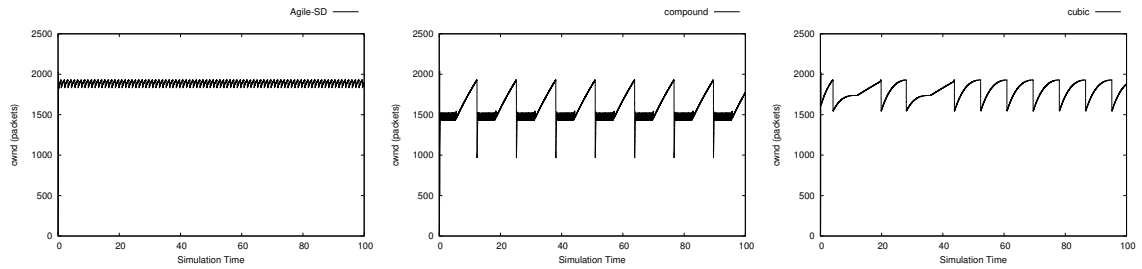
(b) Buffer Size = 25 Packets.



(c) Buffer Size = 100 Packets.

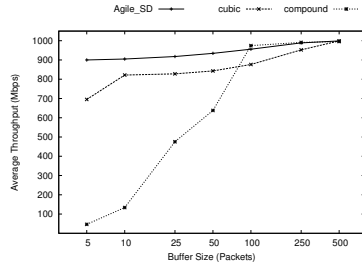


(d) Buffer Size = 250 Packets.

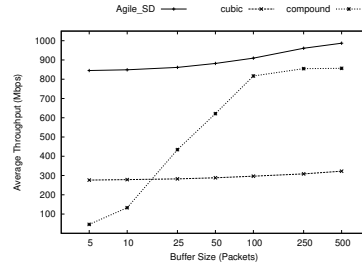


(e) Buffer Size = 500 Packets.

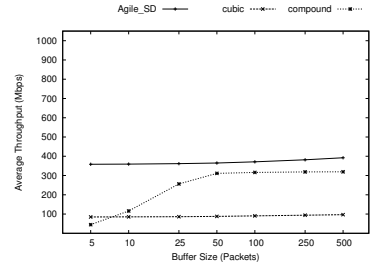
Figure 10: TCP Congestion Window Evolution.



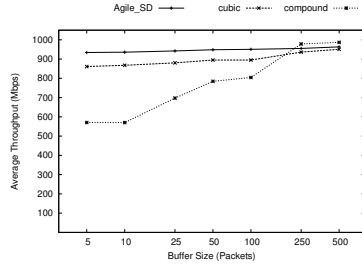
(a) The First Scenario: *Zero* PER.



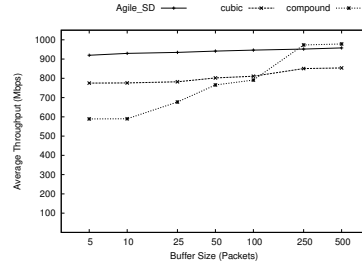
(b) The First Scenario: 10^{-5} PER.



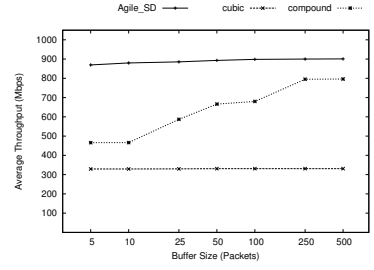
(c) The First Scenario: 10^{-4} PER.



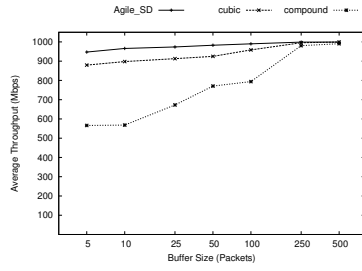
(d) The Second Scenario: *Zero* PER.



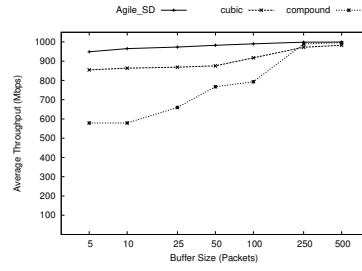
(e) The Second Scenario: 10^{-5} PER.



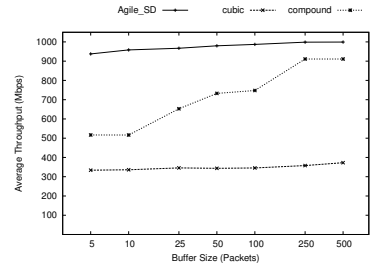
(f) The Second Scenario: 10^{-4} PER.



(g) The Third Scenario: *Zero* PER.



(h) The Third Scenario: 10^{-5} PER.



(i) The Third Scenario: 10^{-4} PER.

Figure 11: The Average Throughput vs. Buffer Size.